Cloud Computing Lab Record

Team members:
1934019 KISHORE S
1934024 NAREAN KARTHICK P
1934046 SREEKANTH S

Exercise 1 **Installing Oracle VirtualBox**

1. Go to https://www.virtualbox.org/wiki/Downloads to download Oracle virtual box.

Select your OS

2. Run the file
3. Click Next

4. Leave the default settings and click next to install

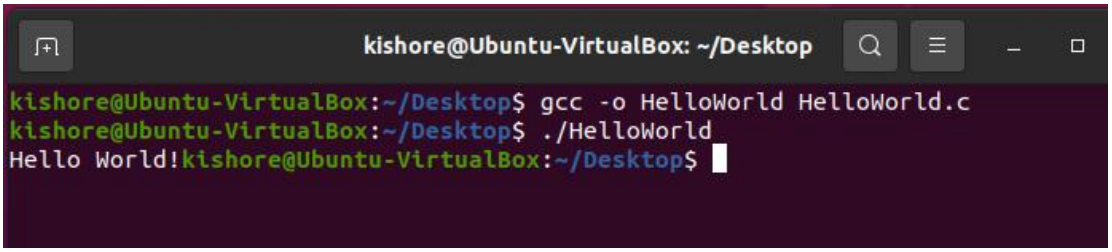
Exercise 2 **Running a C program on Virtual Machine**

Linux

1. Open text editor in Ubuntu VM.
2. Write the helloworld.c Program

```
#include <stdio.h>
void main()
{
        printf("Hello World!");
}
```

3. Save the file.
4. Compile the file using “gcc -o HelloWorld HelloWorld.c”
5. To run the file Type “./HelloWorld”

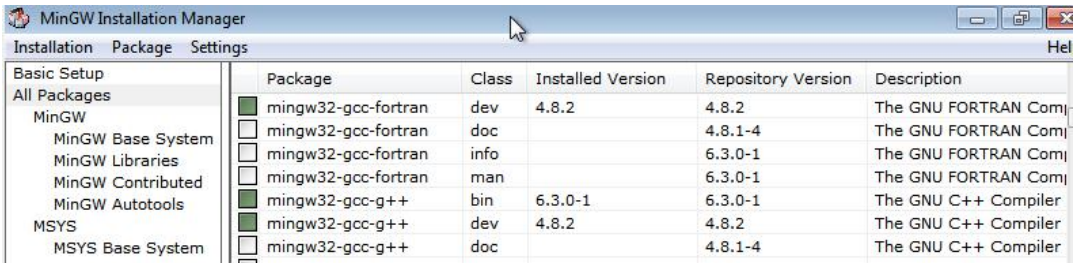


Windows

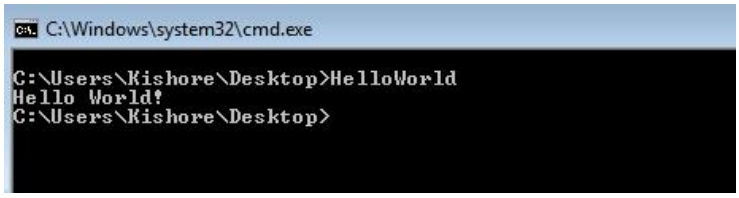
1. Download MinGW GCC Compiler from https://sourceforge.net/projects/mingw/

2. Install C compiler from MinGW

3. In a Text Editor Write the same HelloWorld.c Program

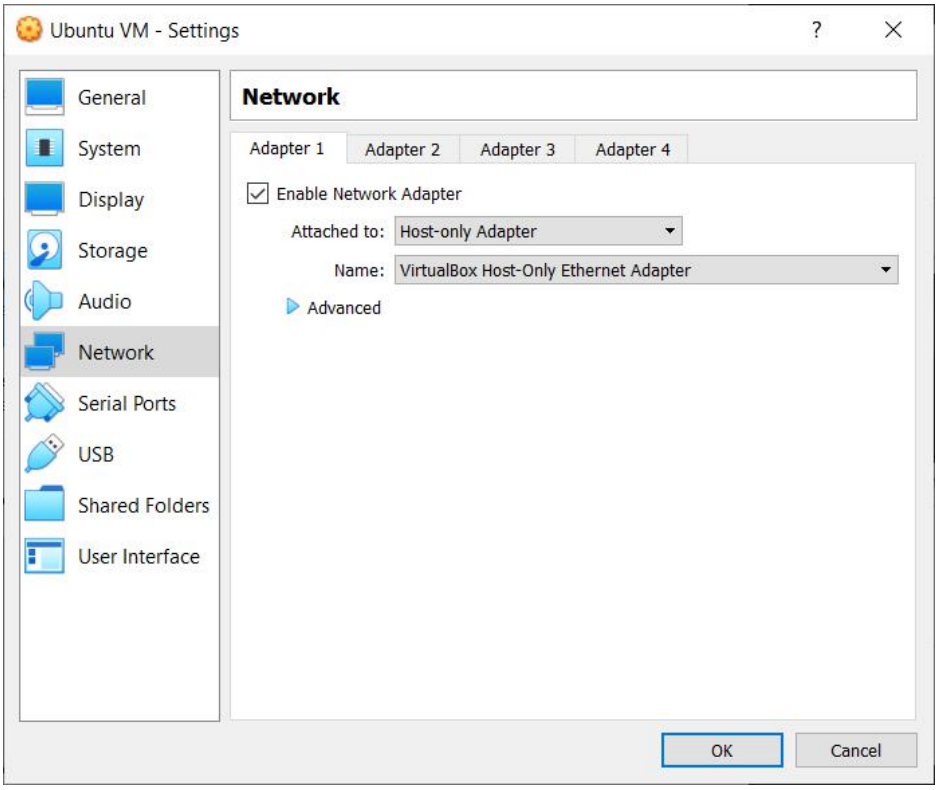


4. In Command Prompt type “gcc -o HelloWorld HelloWorld.c” to compile the program
5. To run the program type “HelloWorld” in Command prompt

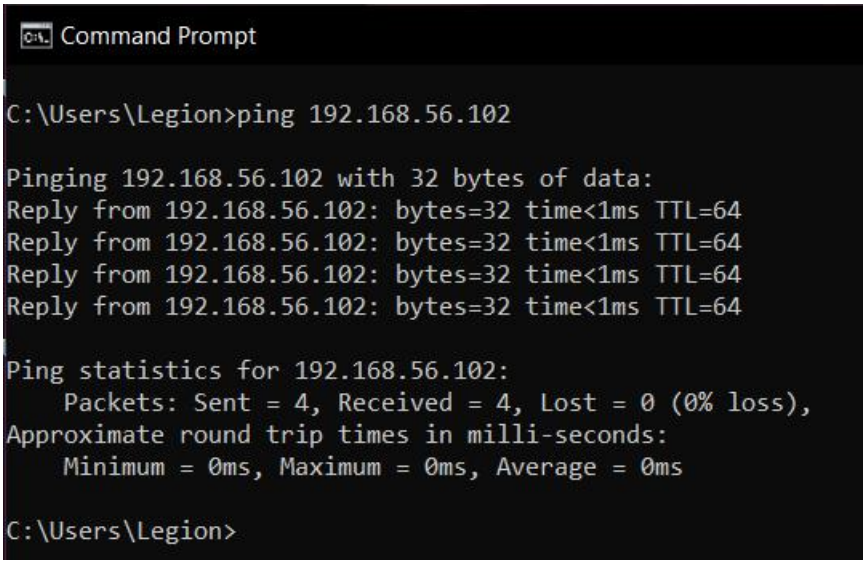


Exercise 3 Communication Between Host and Virtual Machine

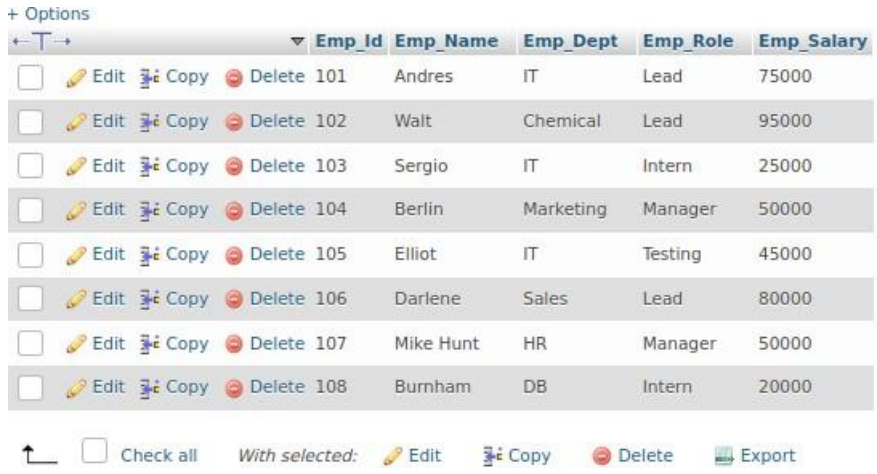
1. Change the adapter Settings to Host-only Adapter



2. Ping the Guest VM from the host



3. Install XAMPP and change the required files
4. Start XAMPP and Create a Database and a Table



+ Options

	Emp_Id	Emp_Name	Emp_Dept	Emp_Role	Emp_Salary
Edit Copy Delete	101	Andres	IT	Lead	75000
Edit Copy Delete	102	Walt	Chemical	Lead	95000
Edit Copy Delete	103	Sergio	IT	Intern	25000
Edit Copy Delete	104	Berlin	Marketing	Manager	50000
Edit Copy Delete	105	Elliot	IT	Testing	45000
Edit Copy Delete	106	Darlene	Sales	Lead	80000
Edit Copy Delete	107	Mike Hunt	HR	Manager	50000
Edit Copy Delete	108	Burnham	DB	Intern	20000

Check all *With selected:* Edit Copy Delete Export

5. Create a new user with IP as % or the IP of guest OS found using ifconfig.

IP

```
enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.102  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::48c:ac8:d986:ba89  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:e5:5d:05  txqueuelen 1000  (Ethernet)
        RX packets 46  bytes 8165 (8.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 48  bytes 5784 (5.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

User

	User name	Host name	Password	Global privileges	User group	Grant	Action
	Any	%	No	USAGE		No	Edit privileges Export
	Any	localhost	No	USAGE		No	Edit privileges Export
	kishore_host	192.168.56.1	Yes	ALL PRIVILEGES		Yes	Edit privileges Export
	kishore_test	%	Yes	ALL PRIVILEGES		Yes	Edit privileges Export
	kk_2	192.168.1.8	Yes	ALL PRIVILEGES		Yes	Edit privileges Export
	pma	localhost	No	USAGE		No	Edit privileges Export
	root	127.0.0.1	No	ALL PRIVILEGES		Yes	Edit privileges Export
	root	::1	No	ALL PRIVILEGES		Yes	Edit privileges Export
	root	localhost	No	ALL PRIVILEGES		Yes	Edit privileges Export

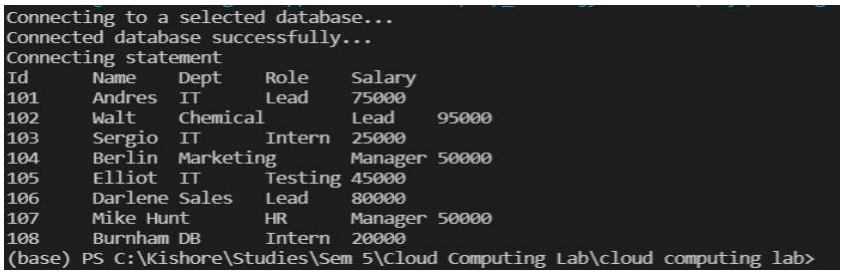
6. Download the MySQL-connector jar file and add it to the project path in Host.

7. Write a Java program to access the database in Guest OS.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class HostVMconnection {
   static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
        static final String DB_URL = "jdbc:mysql://192.168.56.102:3306/employee";
        static final String USER = "kishore_test";
        static final String PASSWORD = "password";

        public static void main(String[] args)
                { Connection conn = null;
                Statement stmt = null;
                try {
                        Class.forName(JDBC_DRIVER);
                        System.out.println("Connecting to a selected database...");
                        conn = DriverManager.getConnection(DB_URL, USER,
PASSWORD);
                        System.out.println("Connected database successfully...");

                        System.out.println("Connecting statement");
                        stmt = conn.createStatement();
                        System.out.println("Id\tName\tDept\tRole\tSalary");
                        String sql = "SELECT
Emp_Id,Emp_Name,Emp_Dept,Emp_Role,Emp_Salary FROM Employee";
                        ResultSet rs = stmt.executeQuery(sql);
                        while(rs.next()) {
                                int id = rs.getInt("Emp_Id");
                                String name = rs.getString("Emp_Name");
                                String dept = rs.getString("Emp_Dept");
                                String role = rs.getString("Emp_Role");
                                int salary = rs.getInt("Emp_Salary");
                                System.out.println(id + "\t" + name + "\t" + dept + "\t" + role +
"\t" + salary);
```

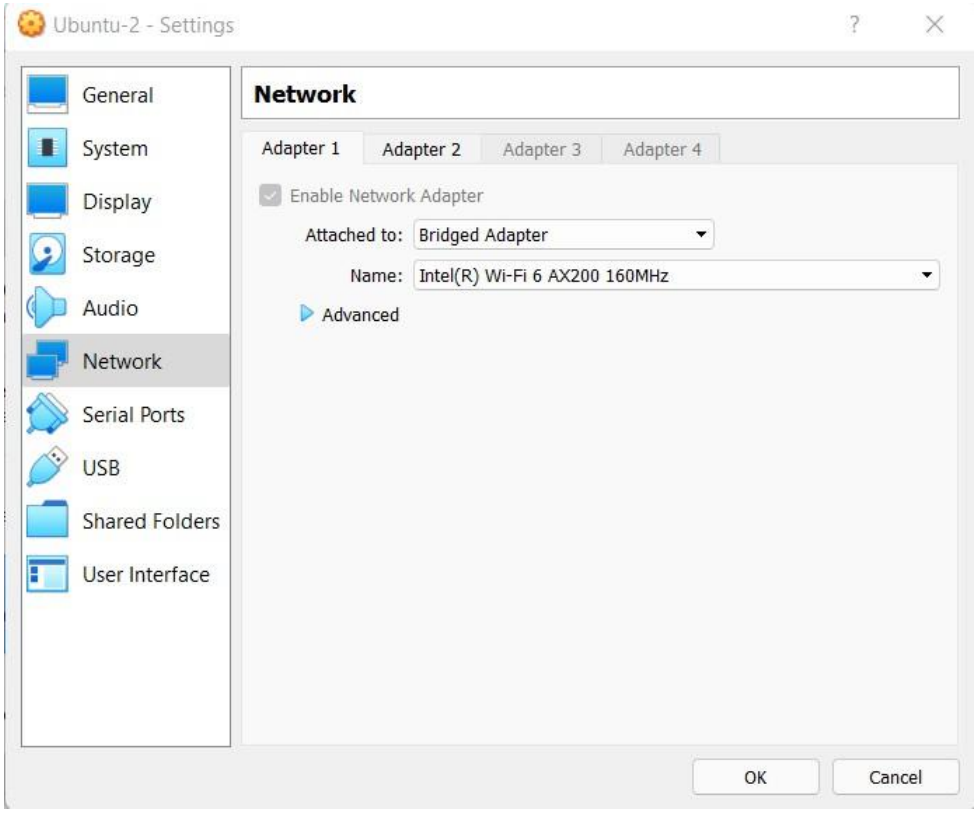
```
                }
                rs.close();
                }
                        catch(SQLException se) {
                se.printStackTrace();
                        }catch(Exception e) {
                                e.printStackTrace();
                        }finally {
                                try {
                                        if(stmt!=null)
                                                conn.close();
                                }catch(SQLException se) {
                        }
                        }try {
                                if(conn!=null)
                                        conn.close();
                        }catch(SQLException se)
                                { se.printStackTrace()
                                ;
                        }
                }
}
```

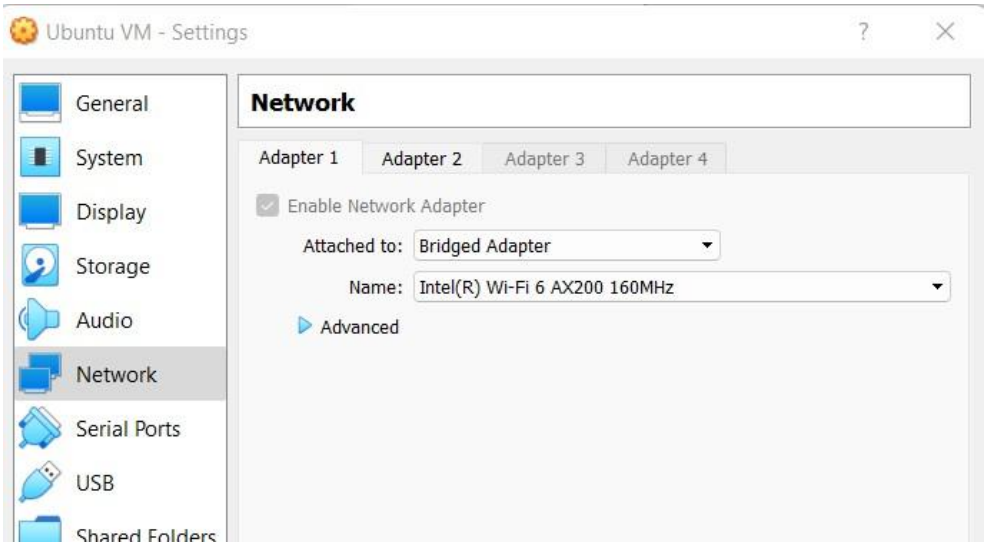
8. Run the program and Verify the result



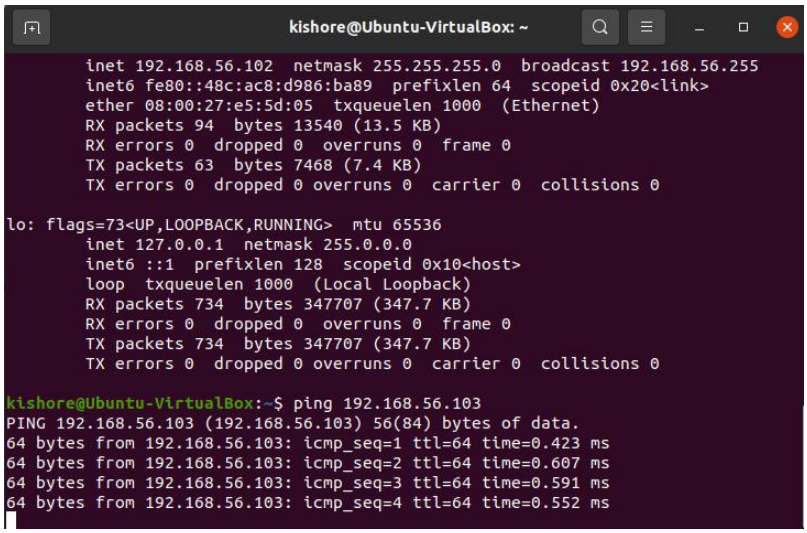
Exercise 4 **VM to VM Connection**

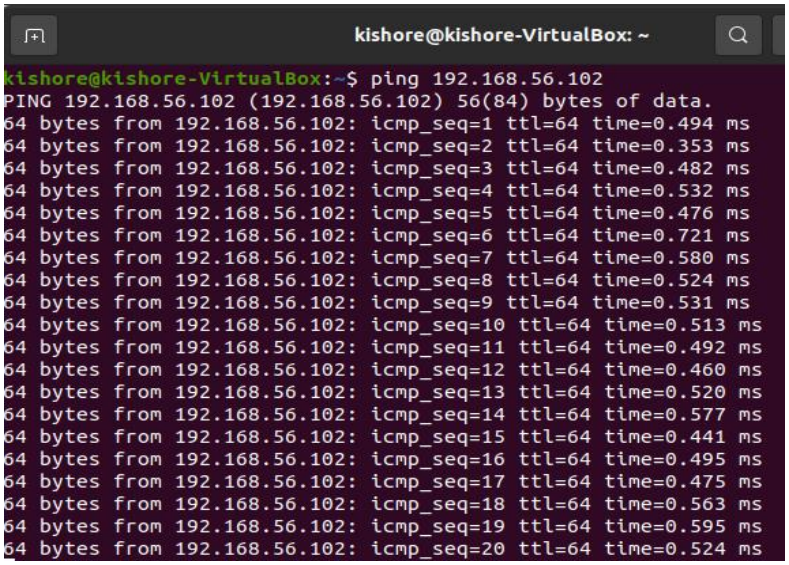
1. After installing two VMs, change their network settings to BridgedAdapter



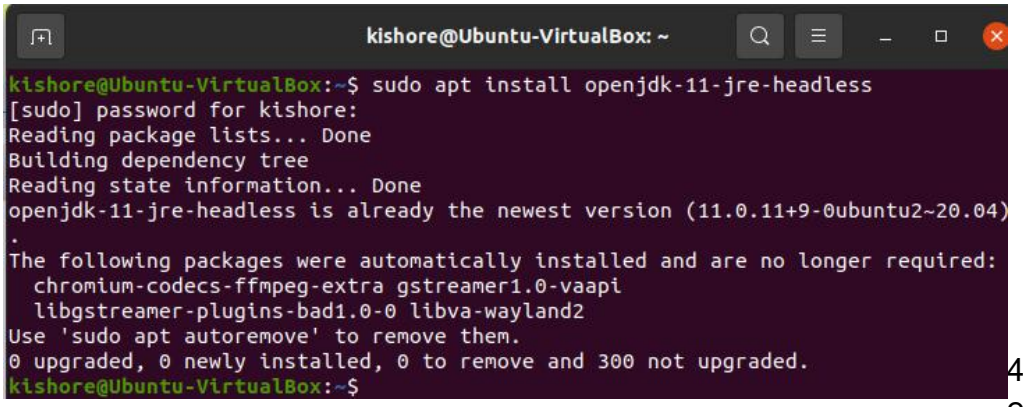


2. Ping and verify both are able to communicate with each other.



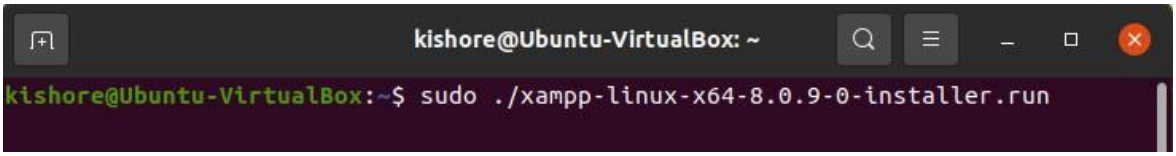


3. Install Java in the VM-B (From which we are going to access the database).

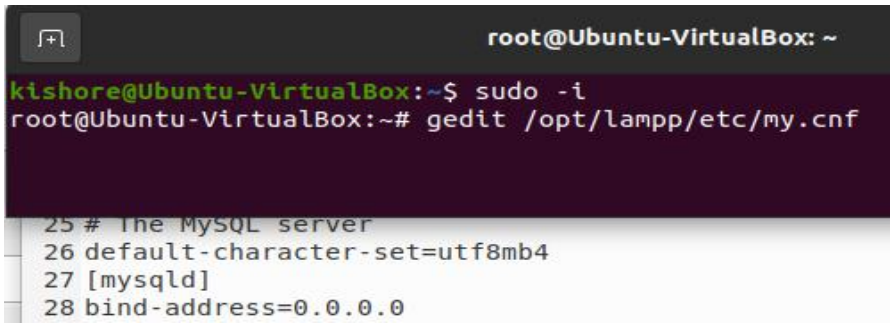


4. Install a code editor/IDE in the VM-B like VisualStudio Code or Eclipse

5. Download the XAMPP package in the VM-A (Where we are going to create the Database and Table).



6. Edit my.cnf file and start XAMPP

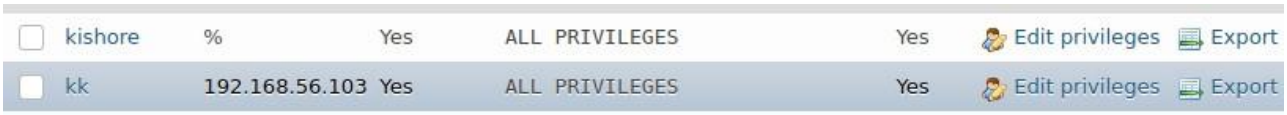


7. Create a Database and Table using phpmyadmin
8. Insert records into the table

+ Options

name	id	role	salary
subash	10	webdev	50000
kishore	11	analyst	35000
jeeva	13	manager	55000
sanjay	14	lead	60000

9. In both the VMs change the second network adapter to Host-only-adapter
10. Create a user in the PHPMyAdmin with IP either as “%” or the IP of the VM-B



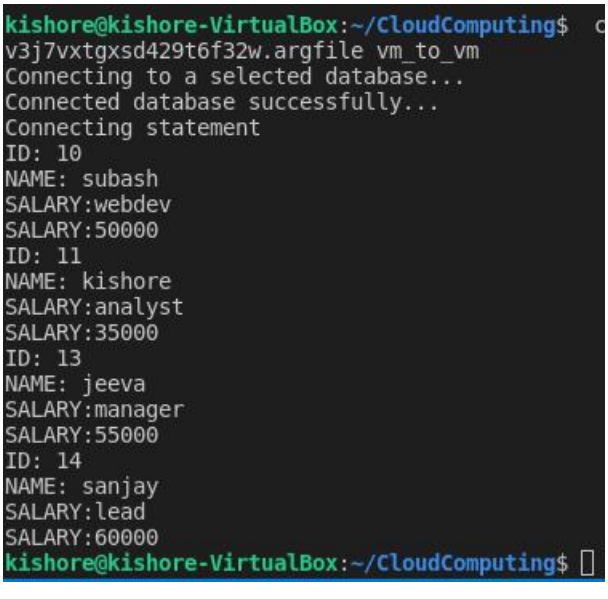
11. Download the MySQL-connector-java and add it to the project path where the code is present in VM-B

12. Write the Java program in VM-B to access the database from VM-A

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class vm_to_vm {
	static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
	static final String DB_URL = "jdbc:mysql://192.168.56.102:3306/Employee-1";
	static final String USER = "kishore";
	static final String PASSWORD = "password";

	public static void main(String[] args) {

		Connection conn = null;
		Statement stmt = null;
		try {
			Class.forName(JDBC_DRIVER);
			System.out.println("Connecting to a selected database...");
			conn = DriverManager.getConnection(DB_URL, USER,
PASSWORD);
			System.out.println("Connected database successfully...");

			System.out.println("Connecting statement");
			stmt = conn.createStatement();

			String sql = "SELECT id,name,role,salary FROM emp";
			ResultSet rs = stmt.executeQuery(sql);
```

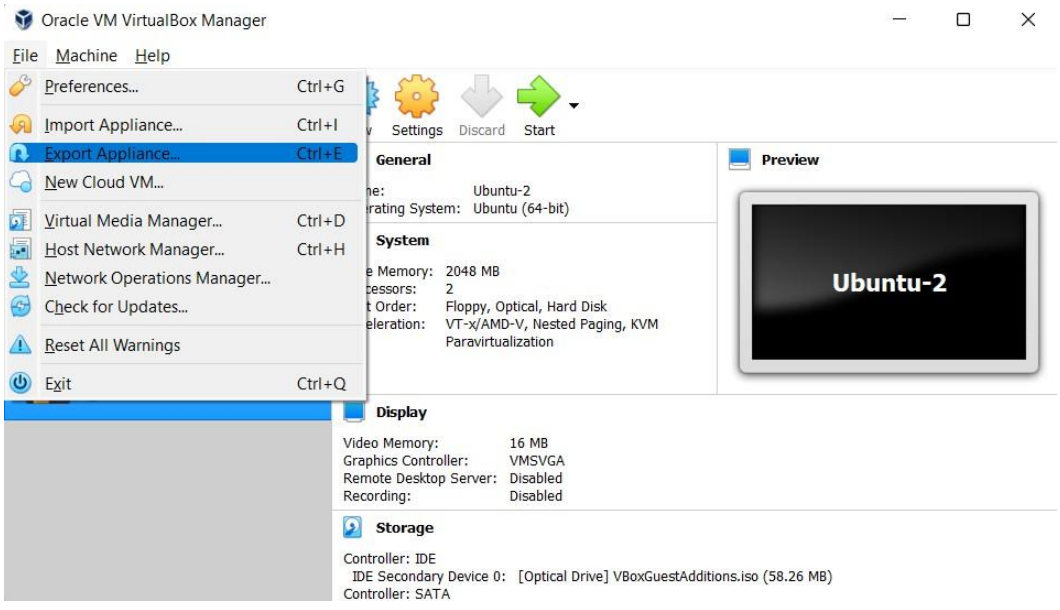
```
while(rs.next()) {
        int id = rs.getInt("id");
        String name = rs.getString("name");
        String role = rs.getString("role");
        int salary = rs.getInt("salary");
        System.out.println("ID: "+id);
        System.out.println("NAME: "+name);
        System.out.println("SALARY:"+role);
        System.out.println("SALARY:"+salary);
}
rs.close();
}
                catch(SQLException se) {
        se.printStackTrace();
                }catch(Exception e) {
                        e.printStackTrace();
                }finally {
                        try {
                                if(stmt!=null)
                                        conn.close();
                        }catch(SQLException se) {
                }
                }try {
                        if(conn!=null)
                                conn.close();
                }catch(SQLException se)
                        { se.printStackTrace()
                        ;
                }
        }
}
```

13. Run the program to see the results

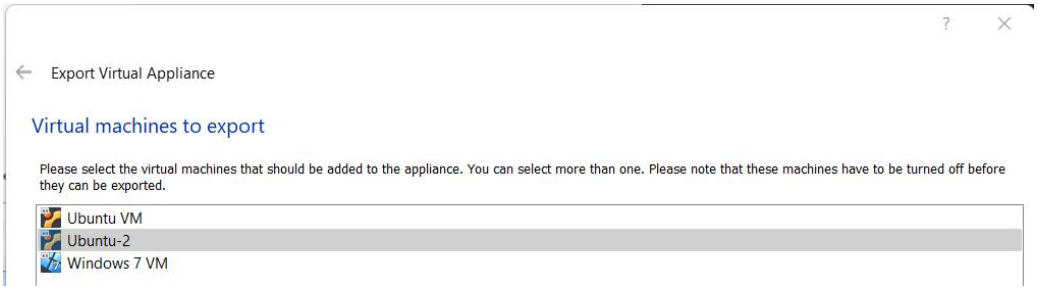


Exercise 5 Exporting VirtualBox VM

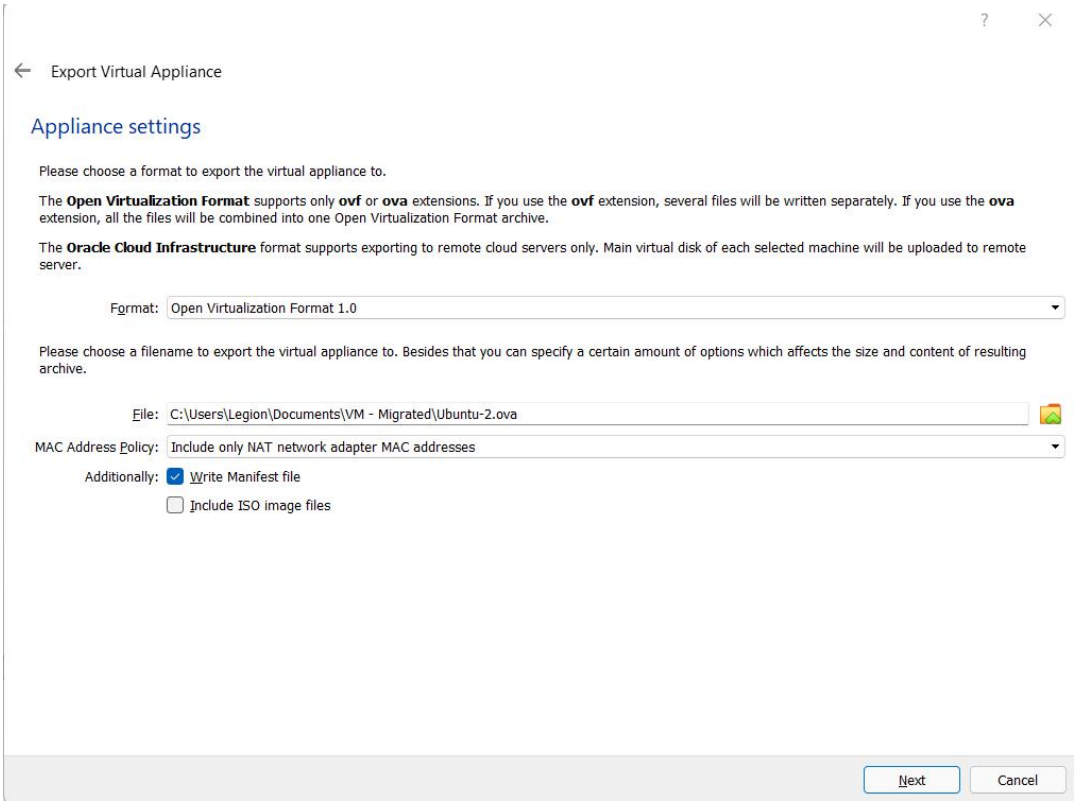
1. Open VirtualBox
2. Click on File --> Export Appliance



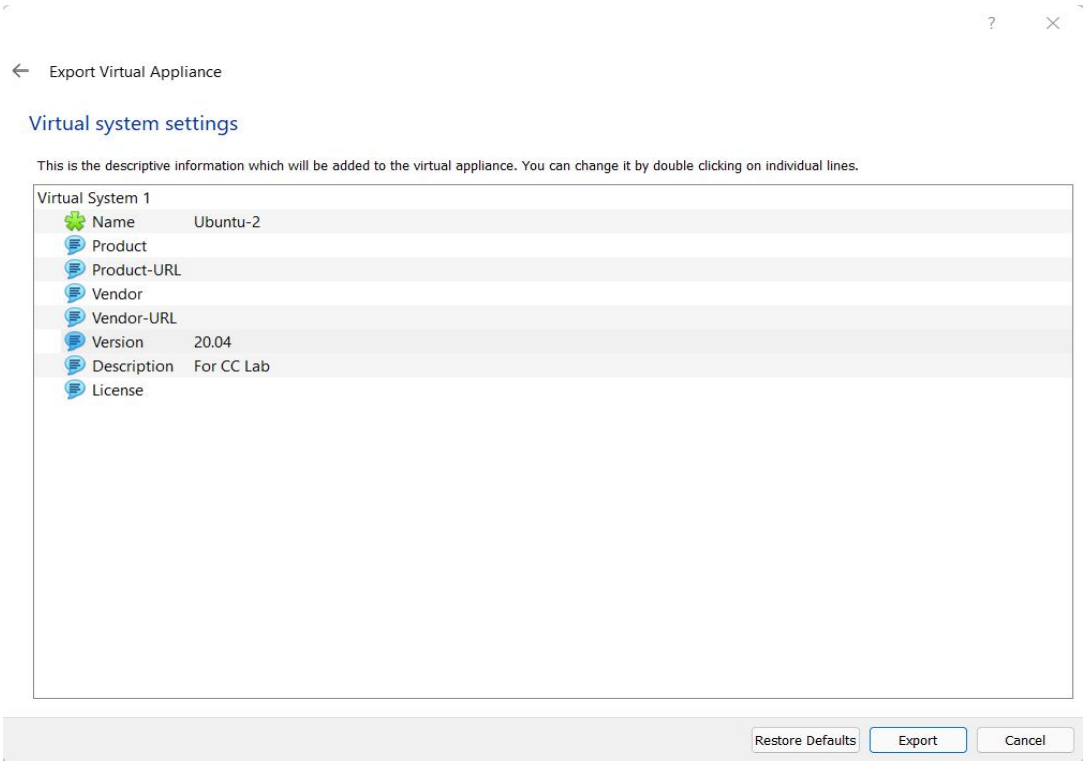
3. Select the VM you want to export



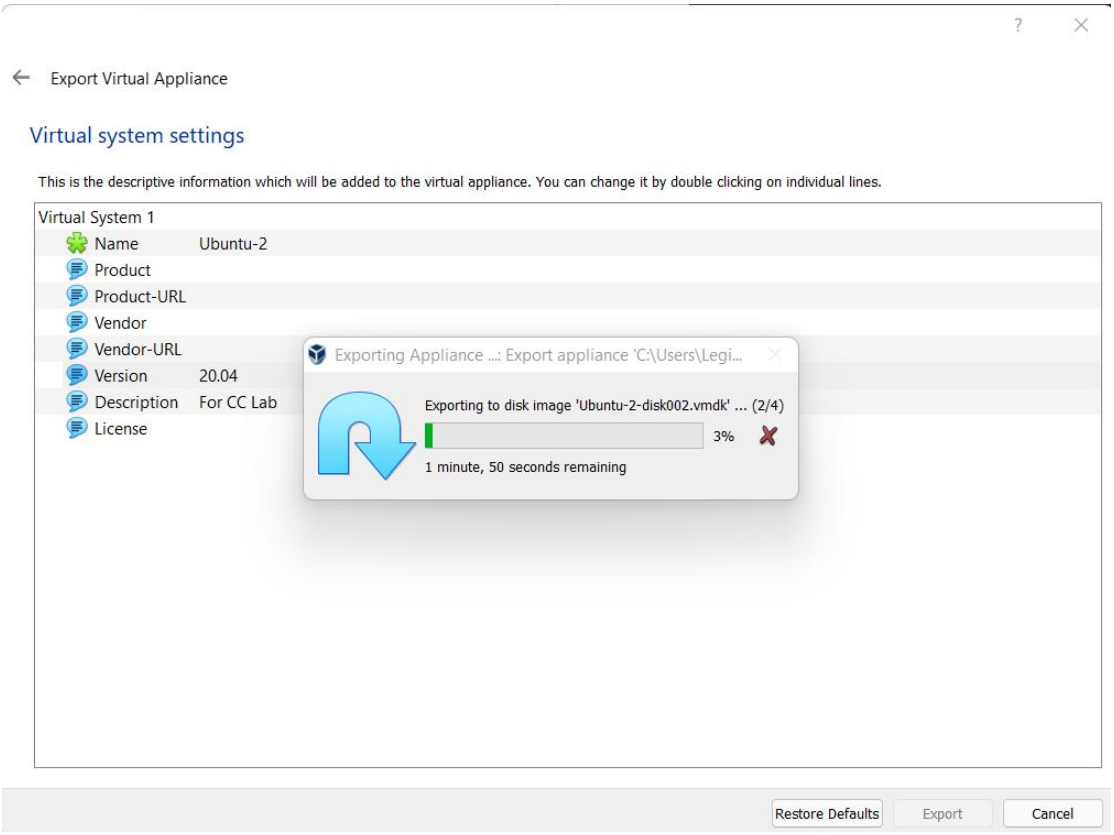
4. Select Location to store the exported VM and click Next



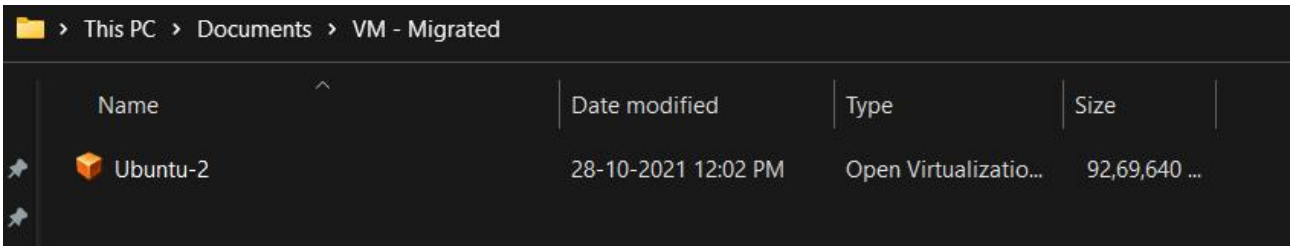
5. Add Description if wanted and click export to store the file in the location specified



6. Wait for the process to complete

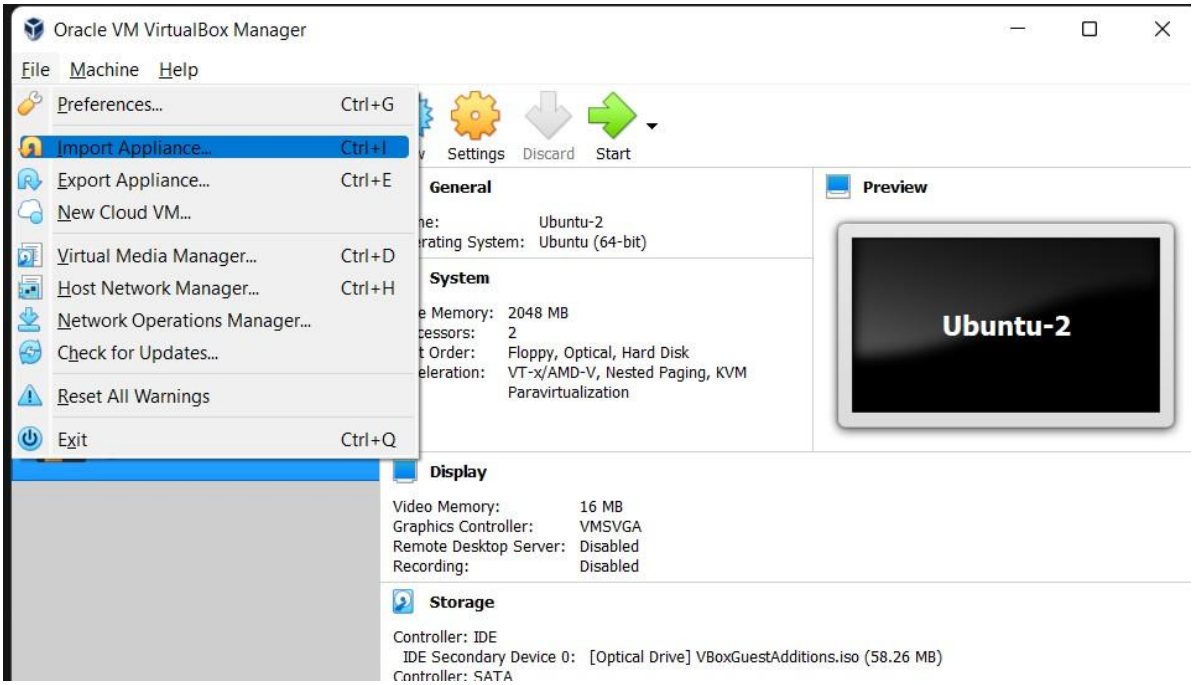


7. Check the Location to see the exported .ova file.

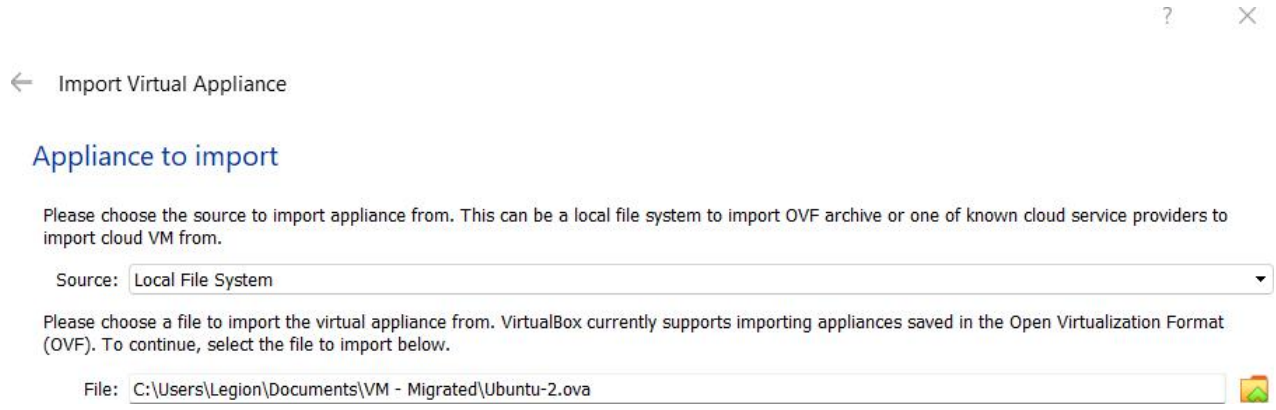


Exercise 6 Importing VirtualBox VM

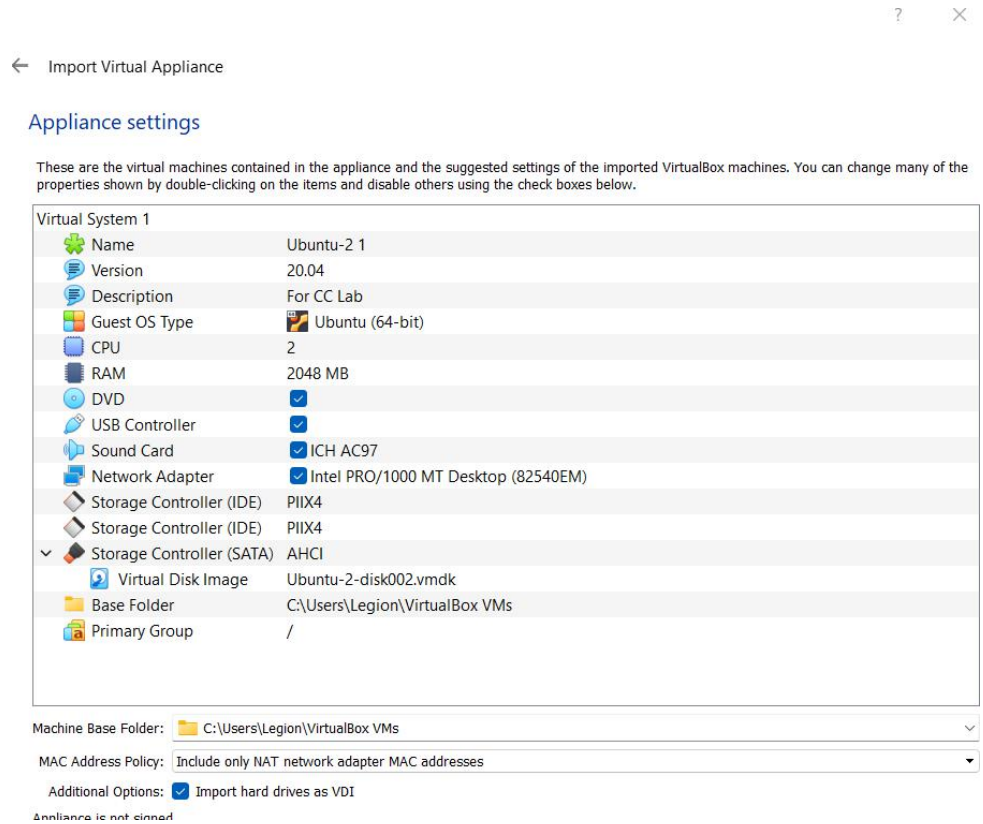
1. Start VirtualBox
2. Click File --> Import Appliance



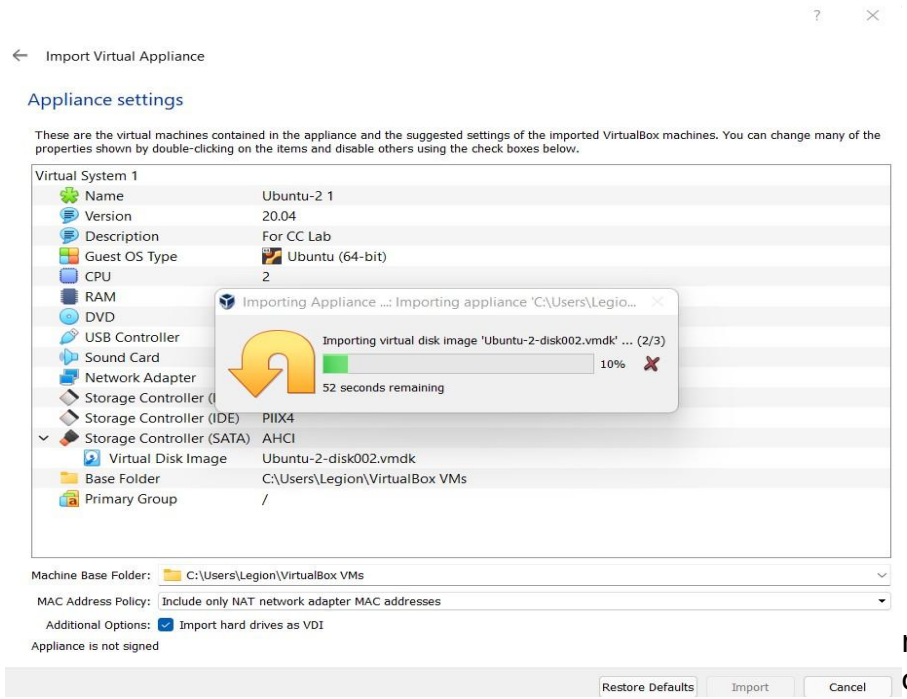
3. Select the OVA file and click next



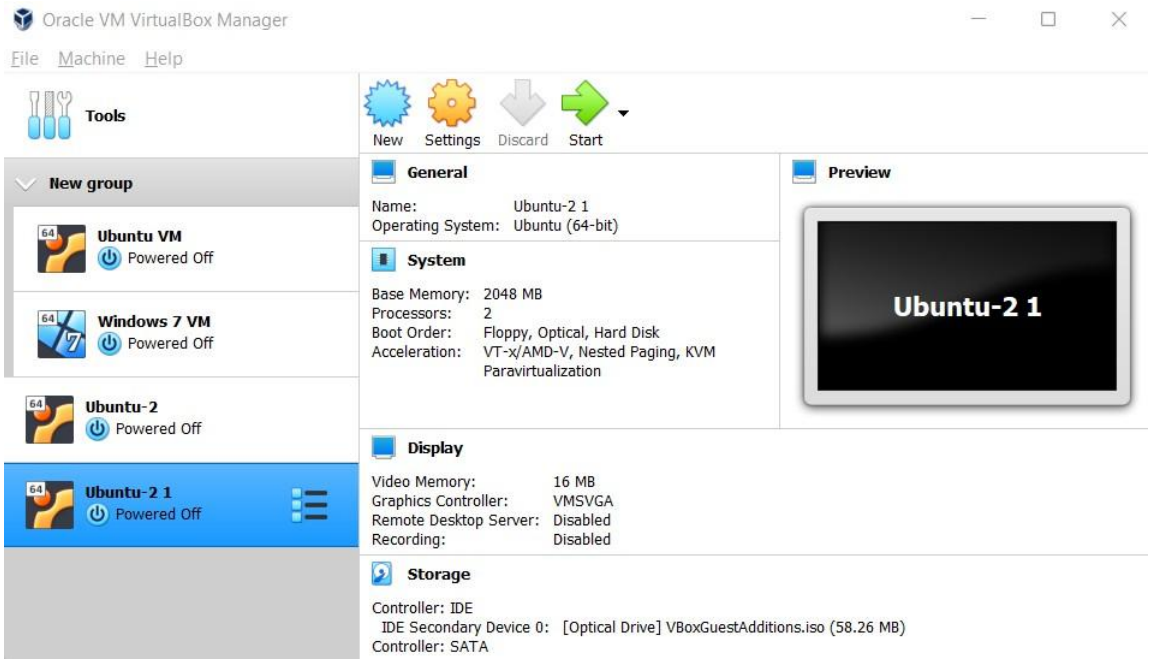
4. Leave the default settings and Click import



5. Wait for the import to finish

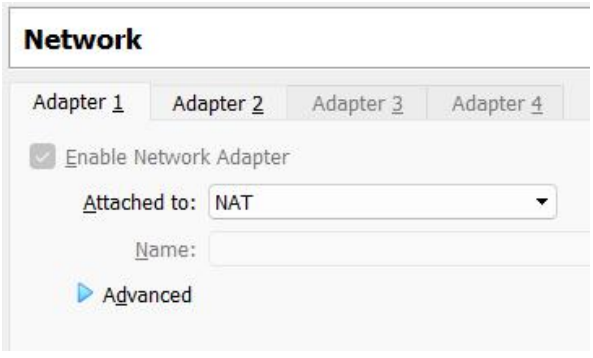


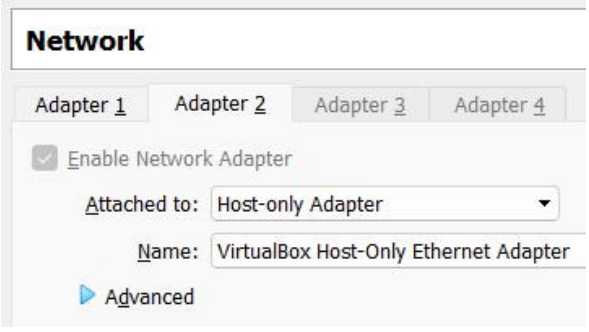
6. The machine will be seen on the VirtualBox homepage



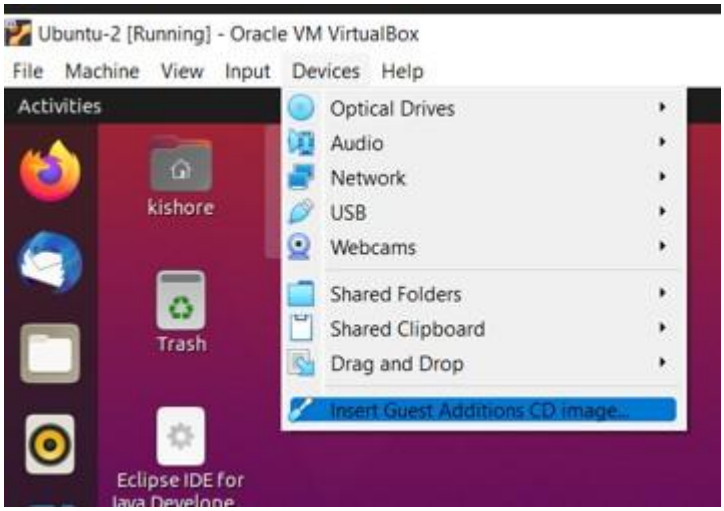
Exercise 7 Creating a shared folder

1. Configuring the network settings

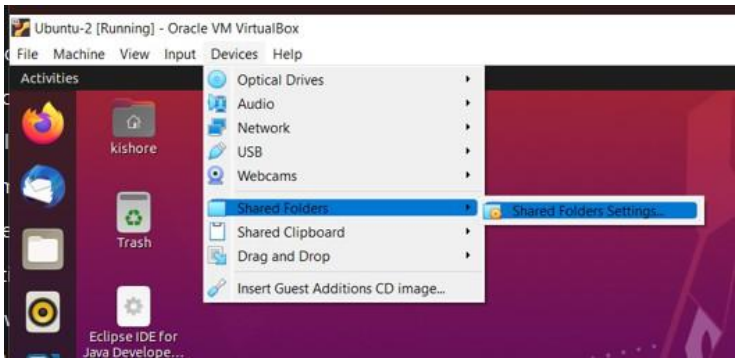




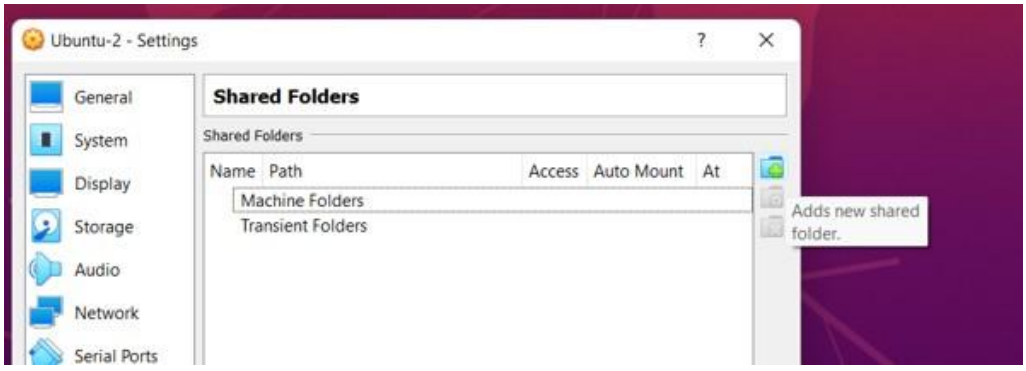
2. Install Guest addition images



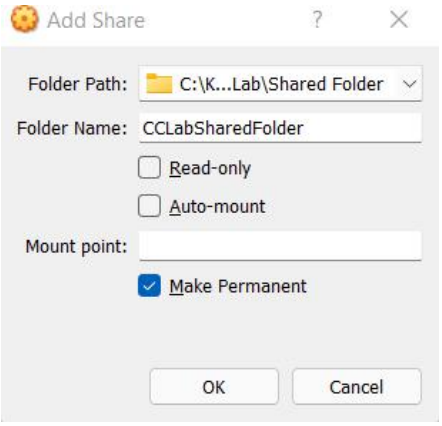
3. Go to shared folder settings



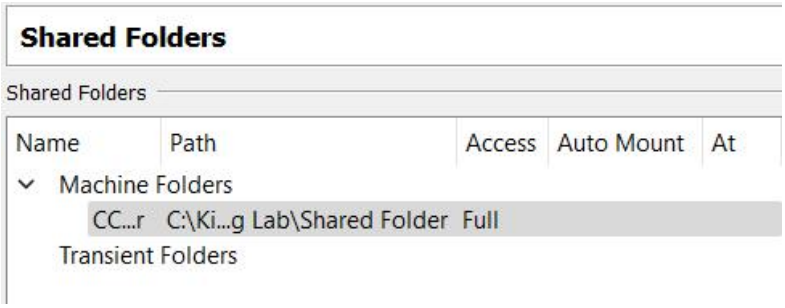
4. Click on Add new Shared folder



5. Select Folder Path and give it a name. You can also make it read-only, auto-mount and permanent by checking the respective boxes



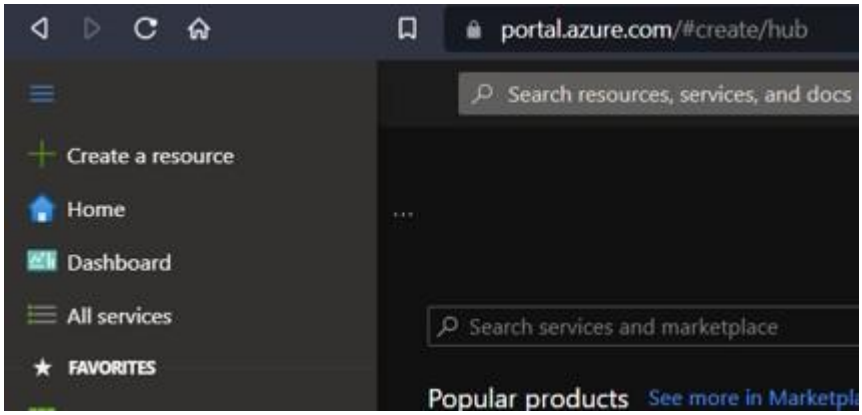
6. Permanent folder will be under machine folders, if not under Transient folder



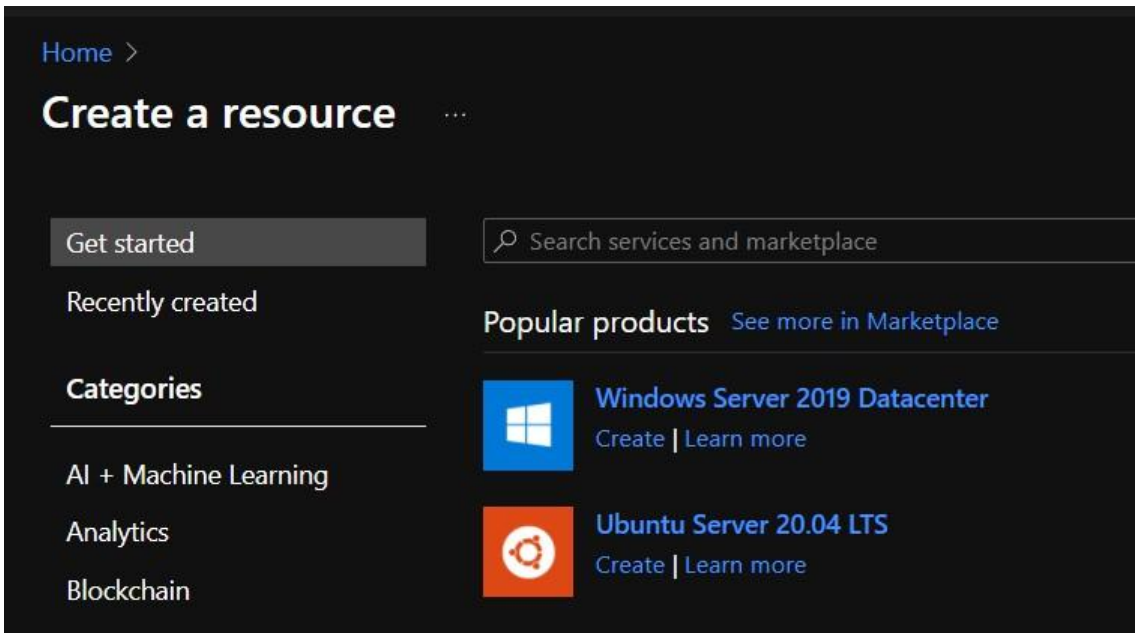
Exercise 8 **Creating a VM on Azure**

1. Click on the three horizontal bars to see the create resource icon.

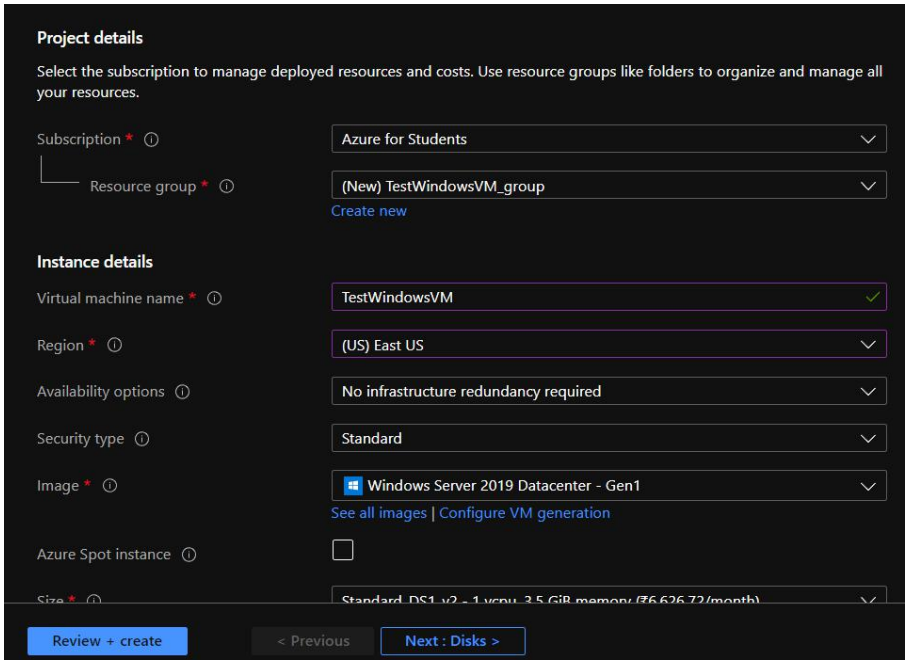
2. Click on create resource



3. Select windows server

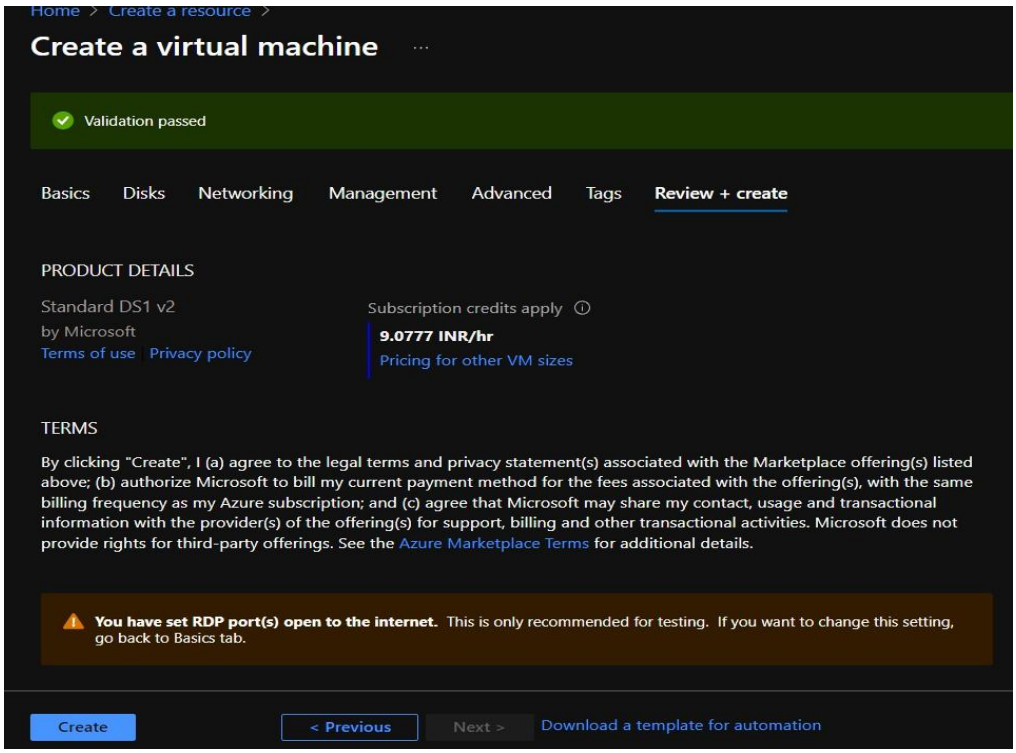


4. Fill out the details

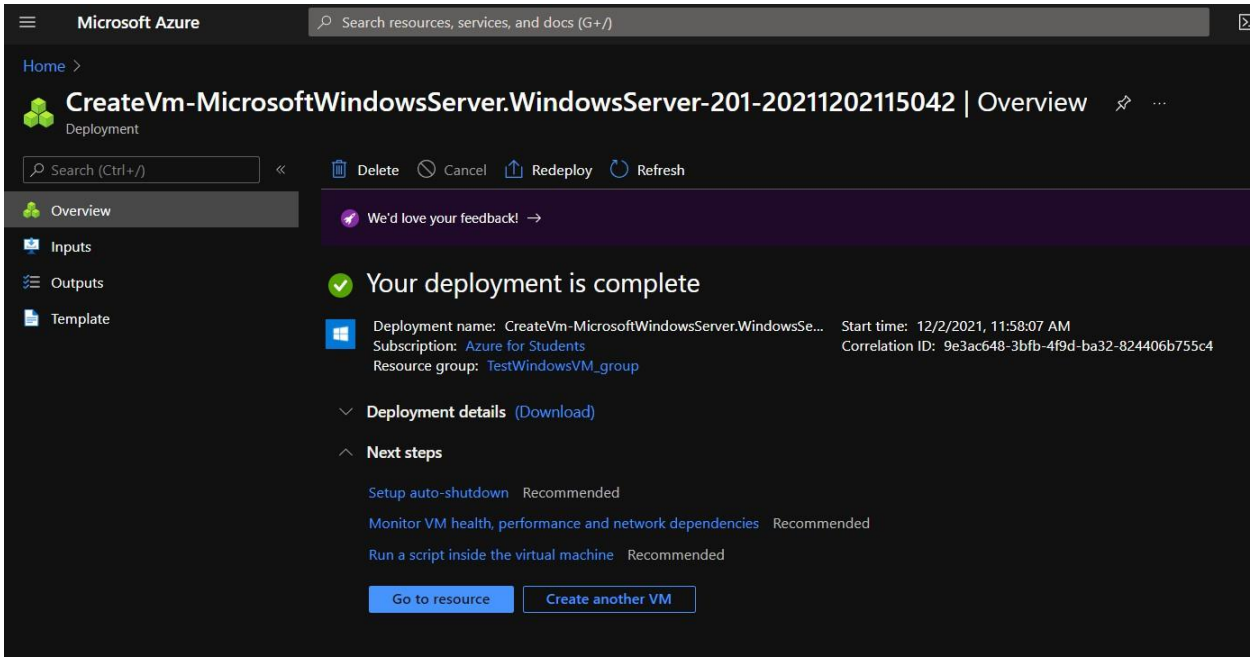


5. Click review and create

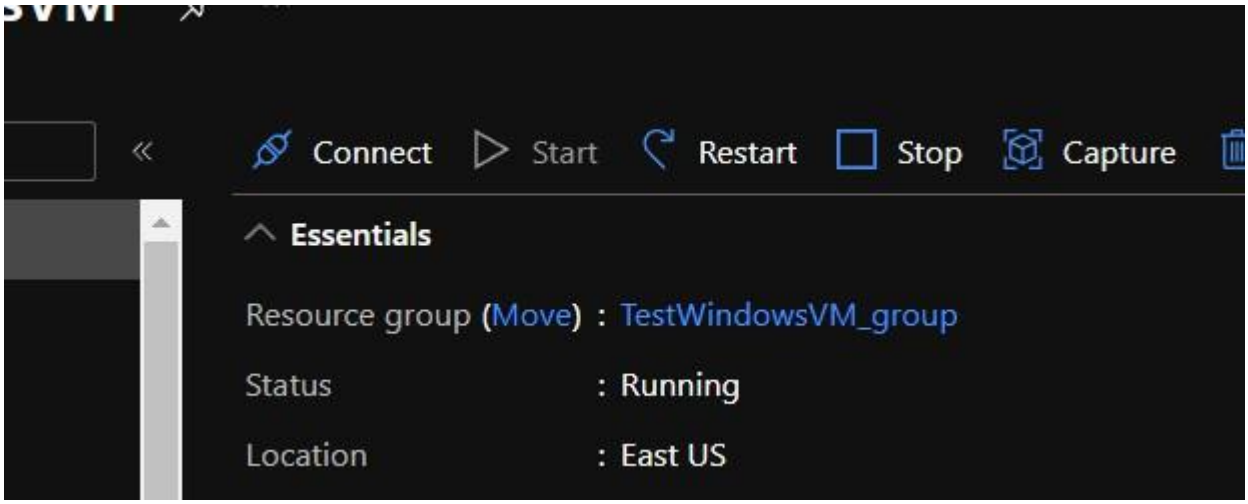
6. Click on create



7. Click on Go to resource once deployed

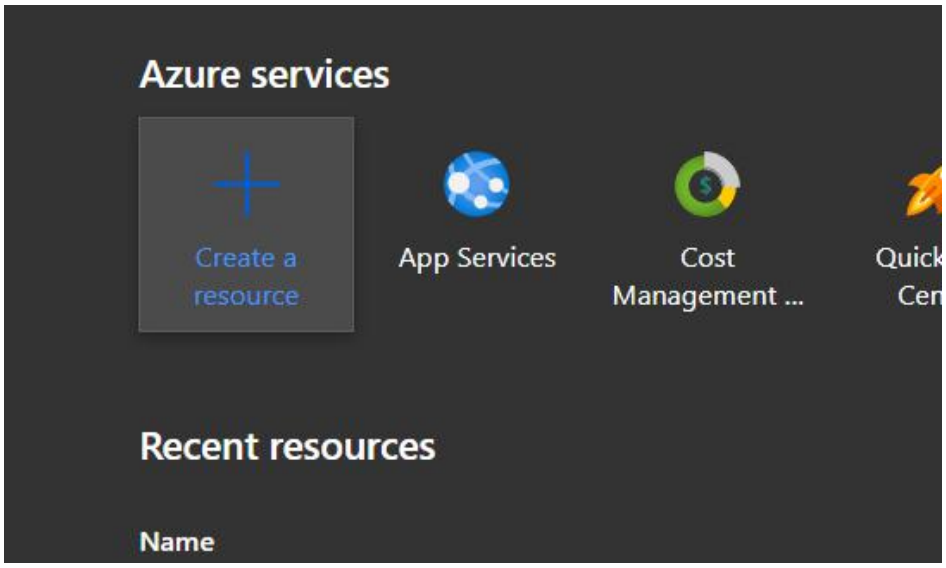


8. Click connect to remotely connect using SSH or RDP (If needed)

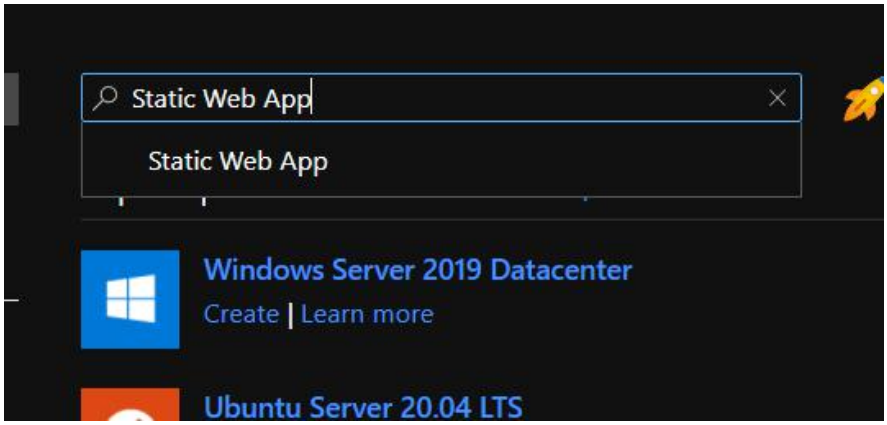


Exercise 9 Creating a static web app in Azure Portal

1. Click on Create Resource

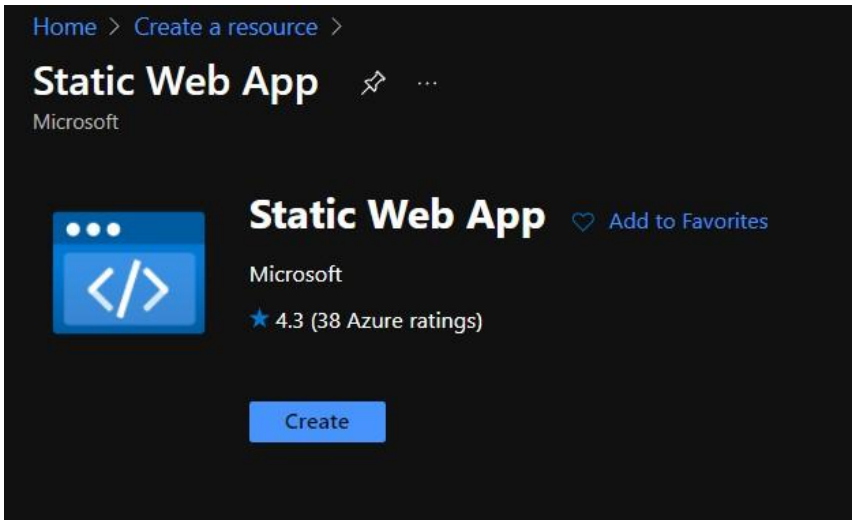


2. Search for Static Web App

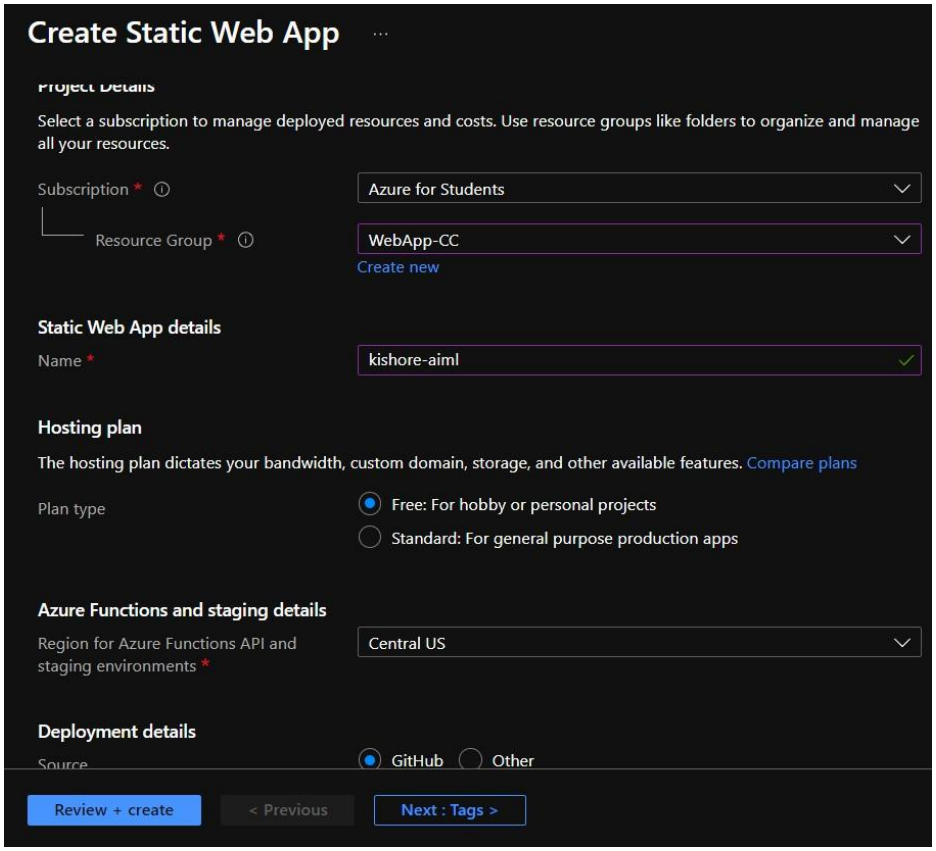


3. Select Static web app

4. Select Create

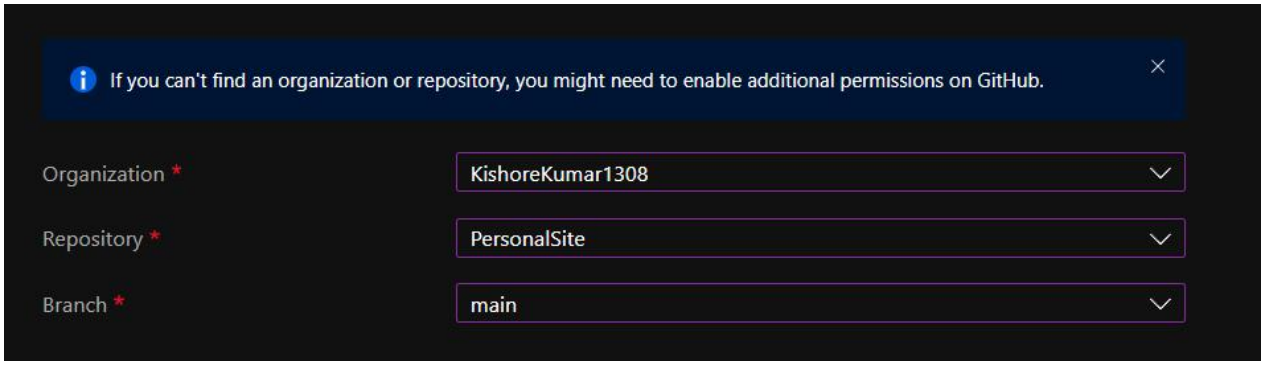


5. Fill out the details

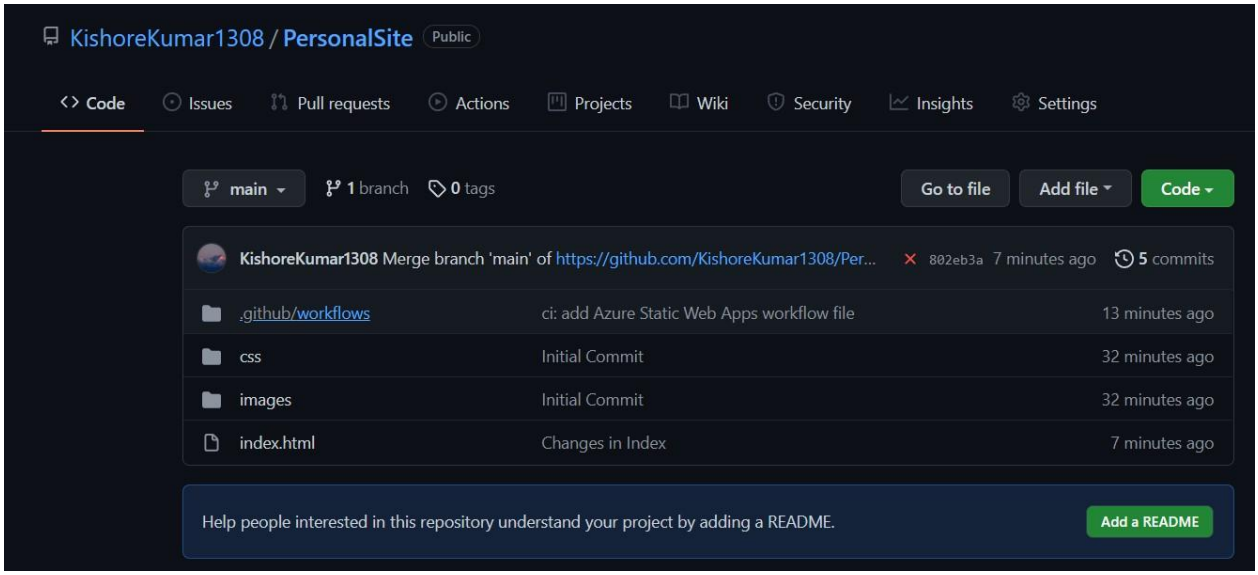


5. Sign in via GitHub

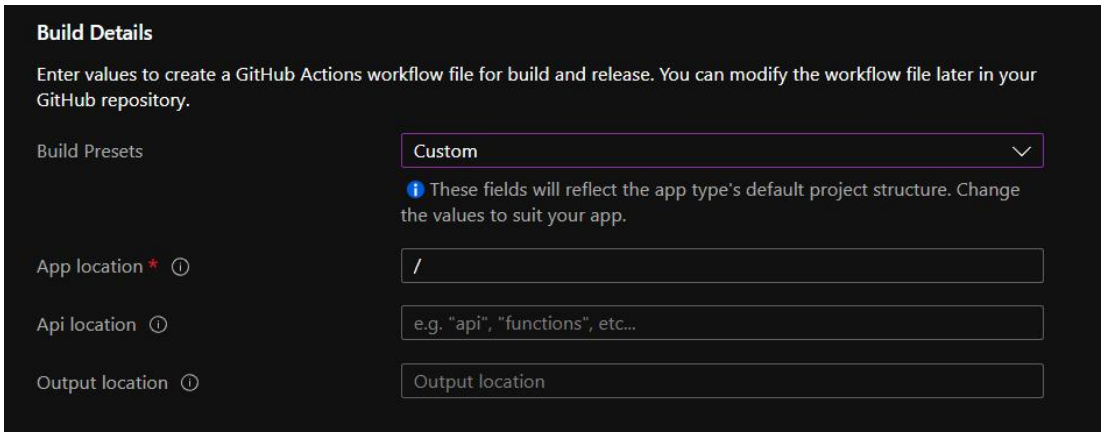
6. Select organization, Repository and Branch where the HTML file is located



7. The GitHub repo should look like this



8. Fill in the following Build Details

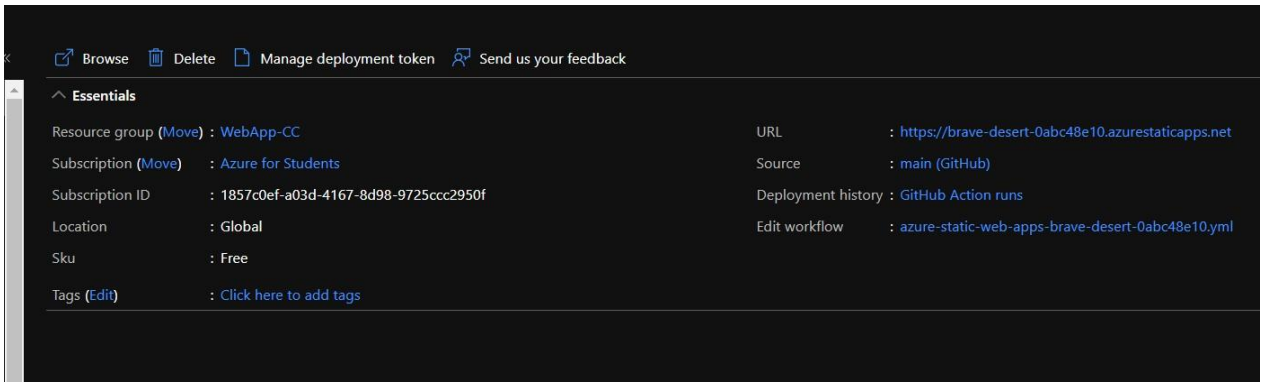


9. Click on Review + Create

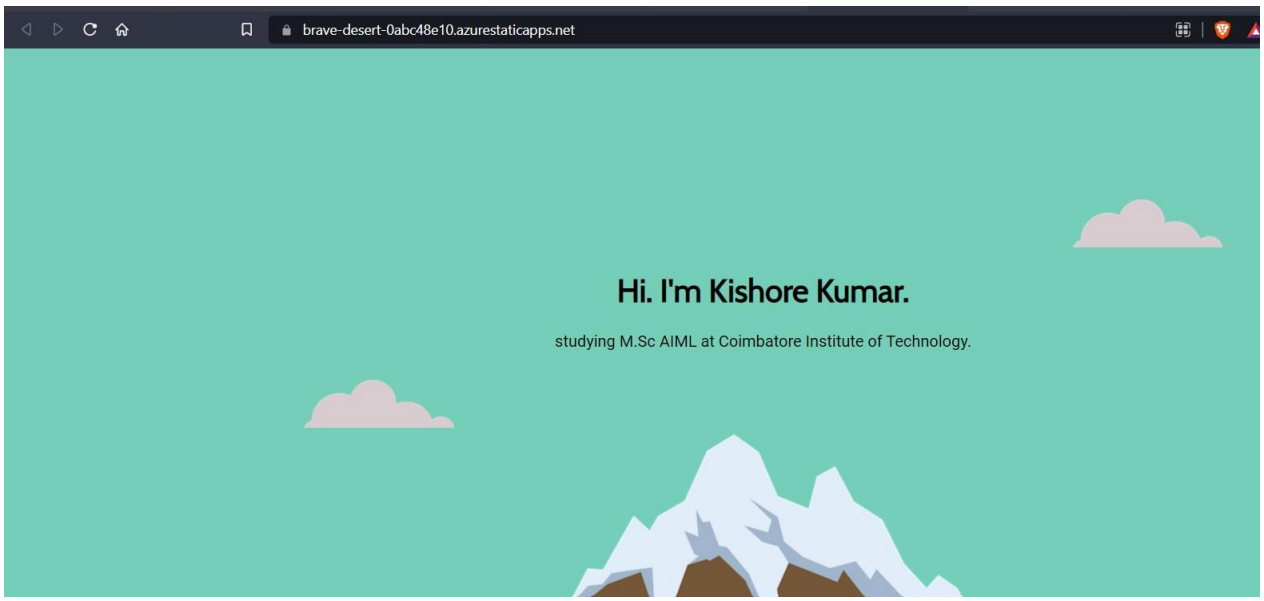
10. Click on Create

11. Click go to resource

12. The URL for the site will be available



13. Click on the URL to see if it is working



14. This is the URL for the Static Web Page created by me:
https://brave-desert-0abc48e10.azurestaticapps.net/

Exercise 10 **Creating a Docker Container**

1. Install a Linux OS in VirtualBox.
2. Install net-tools and ssh.

```
kishore@kishore-VirtualBox:~$ sudo apt install net-tools && ssh
Reading package lists... Done
Building dependency tree
Reading state information... Done
net-tools is already the newest version (1.60+git20180626.aebd88e-1ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-B bind_interface]
           [-b bind_address] [-c cipher_spec] [-D [bind_address:]port]
           [-E log_file] [-e escape_char] [-F configfile] [-I pkcs11]
           [-i identity_file] [-J [user@]host[:port]] [-L address]
           [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
           [-Q query_option] [-R address] [-S ctl_path] [-W host:port]
           [-w local_tun[:remote_tun]] destination [command]
```

3. Find IP of the Linux guest machine.

```
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::217a:e787:ba22:70a5  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:4e:47:be  txqueuelen 1000  (Ethernet)
        RX packets 1624  bytes 707989 (707.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1536  bytes 261203 (261.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

4. In the VirtualBox Setting for this OS, Go to Network -> Advanced -> Port Forwarding, Give Host port as 20022, Guest IP found from ifconfig, and Guest port as 22. Save the changes.

5. In the host machine, Open Command prompt. Type ssh Linux-username@127.0.0.1 -p 20022. Grant access if asked and type the guest Linux OS password.

```
C:\Users\Legion>ssh kishore@127.0.0.1 -p 20022
kishore@127.0.0.1's password:
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.11.0-40-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 updates can be applied immediately.

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Fri Nov 12 16:45:47 2021 from 10.0.2.2
kishore@kishore-VirtualBox:~$
```

6. Create a simple HelloWorld program in the language of your choice. (GoLang here)

```
package main

import (
        "fmt"
)

func main() {
        fmt.Println("Hello, playground")
}

~
```

7. Build the program and verify the output

```
kishore@kishore-VirtualBox: ~/Record
kishore@kishore-VirtualBox:~/Record$ go build helloworld.go
kishore@kishore-VirtualBox:~/Record$ ./helloworld
Hello, playground
kishore@kishore-VirtualBox:~/Record$
```

8. Go to https://docs.docker.com/engine/install/ubuntu/ and add the docker repository to the Ubuntu OS.

Set up the repository

1. Update the `apt` package index and install packages to allow `apt` to use a repository over HTTPS:

```
$ sudo apt-get update

$ sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

2. Add Docker's official GPG key:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyrin
```

3. Use the following command to set up the **stable** repository. To add the **nightly** or **test** repository, add the word `nightly` or `test` (or both) after the word `stable` in the commands below. Learn about **nightly** and **test** channels.

```
$ echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

9. Install Docker

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

10. In Linux create a “Dockerfile” with the following contents

```
Open                                   Dockerfile
                                       ~/Record
1 from golang:latest
2 copy hellowrold.go .
3 cmd go run hellworld.go
```

11. To build the container type “sudo docker build -t test:1 .”
12. To view the image do “sudo docker images”
13. To run the image do “sudo docker run test:1”

```
kishore@kishore-VirtualBox:~$ sudo docker run test:7
Hello, World
```

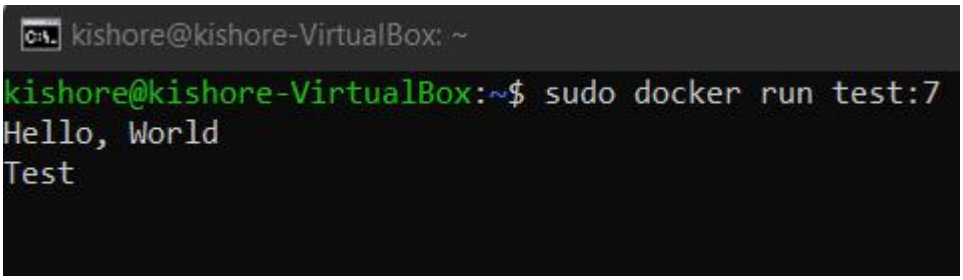
Exercise 11 Entering into Container making Logs of the activities

1. Modify the helloworld program to run for 5 minutes

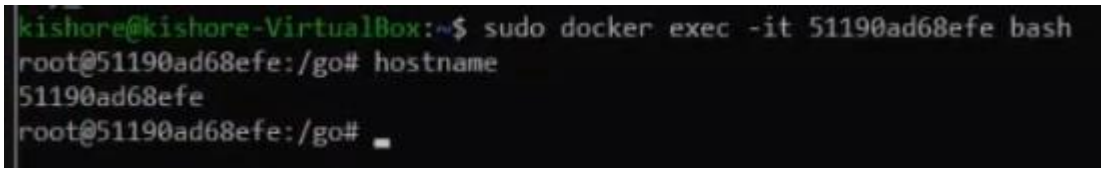
```
Open
1  package main
2
3  import (
4          "fmt"
5          "time"
6  )
7
8  func main() {
9          fmt.Println("Hello, World")
10         fmt.Println("Test")
11         time.Sleep(300 * time.Second)
12 }
13
```

2. Open two SSH terminals from host

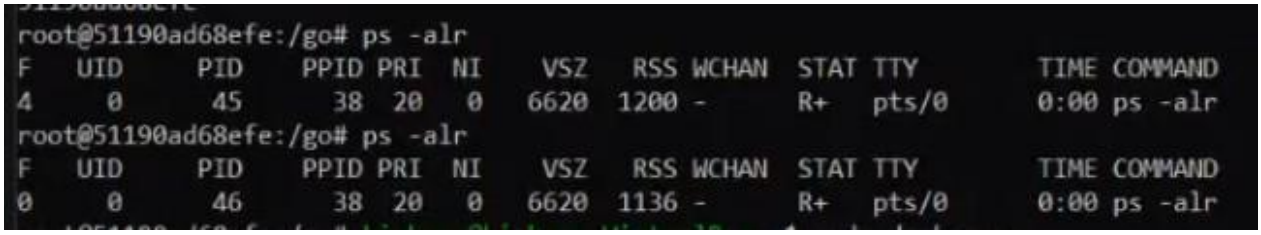
3. Build the modified hellworld program as a docker container and run it in SSH one terminal.



4. Enter into the running container from the other terminal using the following commands



5. To view the process running inside the container do



6. Add log comments to the file to check for any runtime errors

```go
package main

import (
        "fmt"
        "time"
        "log"
)

func main() {
        log.Println("Entering Main Function")
        fmt.Println("Hello, World")
        // select {}
        fmt.Println("Test")
        log.Println("Starting Sleep")
        time.Sleep(300 * time.Second)
        log.Println("Exiting Main")
}
```

7. To view, the log messages do the following

```
kishore@kishore-VirtualBox:~$ sudo docker logs 389688c98834
2021/09/25 06:26:03 Entering Main Function
Hello, World
Test
2021/09/25 06:26:03 Starting Sleep
```

8. Instead of using the entire golang, we can use only the required packages using alpine

Exercise 12 **Installing Kubernetes**

1. Go to https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/
2. Install the prerequisites as mentioned in the website

```
kishore@kishore-VirtualBox:~$ cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
> br_netfilter
> EOF
t <<EOF | sudo tee[sudo] password for kishore:
 /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sudo sysctl --systemSorry, try again.
```

3. To install kubeadm, kubectl, kubelet type the commands as mentioned in the website

Debian-based distributions | Red Hat-based distributions | Without a package manager

1. Update the `apt` package index and install packages needed to use the Kubernetes `apt` repository:

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
```

2. Download the Google Cloud public signing key:

```
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

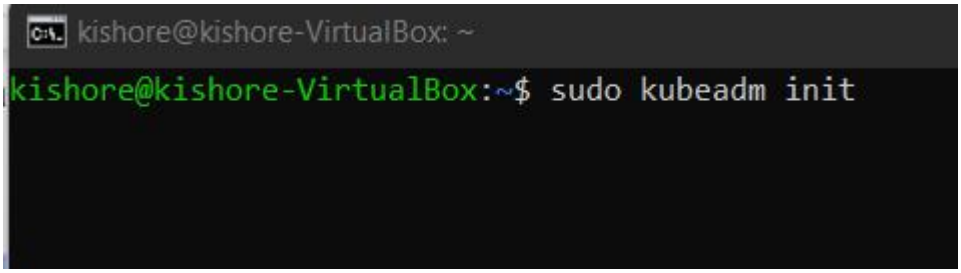
3. Add the Kubernetes `apt` repository:

```
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /e
```

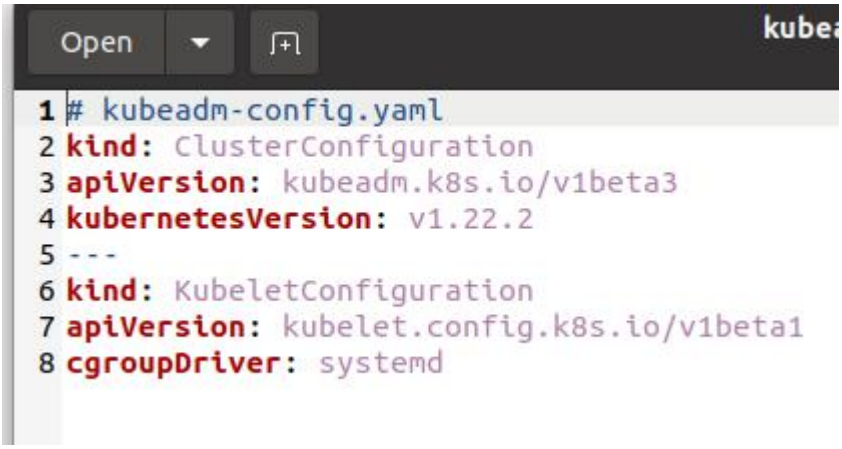
4. Update `apt` package index, install kubelet, kubeadm and kubectl, and pin their version:

```
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

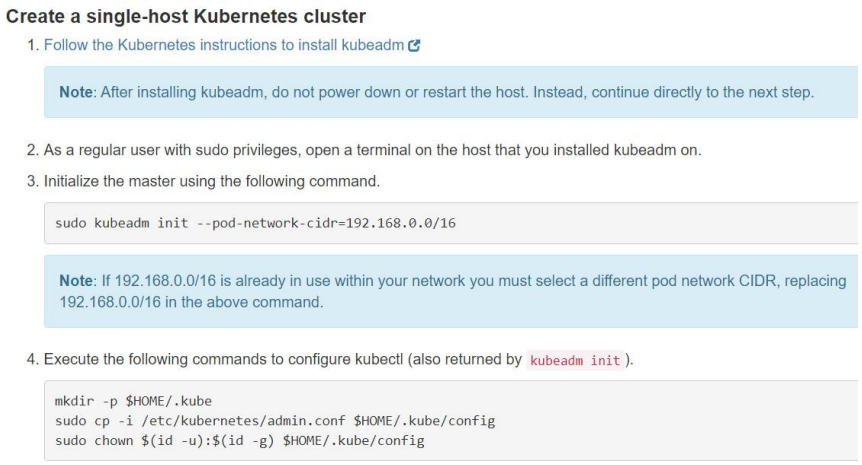
4. To initialize kubeadm, use sudo kubeadm init.



5. Turn off swap using swap -a
6. Create a kube configuration yaml file



7. Use “kubectl get pods -n kube-system” to see the pods.
8. Add Calico for Kuberentes from https://docs.projectcalico.org/getting-started/kubernetes/quickstart



Create a single-host Kubernetes cluster

1. Follow the Kubernetes instructions to install kubeadm

> **Note**: After installing kubeadm, do not power down or restart the host. Instead, continue directly to the next step.

2. As a regular user with sudo privileges, open a terminal on the host that you installed kubeadm on.
3. Initialize the master using the following command.

```
sudo kubeadm init --pod-network-cidr=192.168.0.0/16
```

> **Note**: If 192.168.0.0/16 is already in use within your network you must select a different pod network CIDR, replacing 192.168.0.0/16 in the above command.

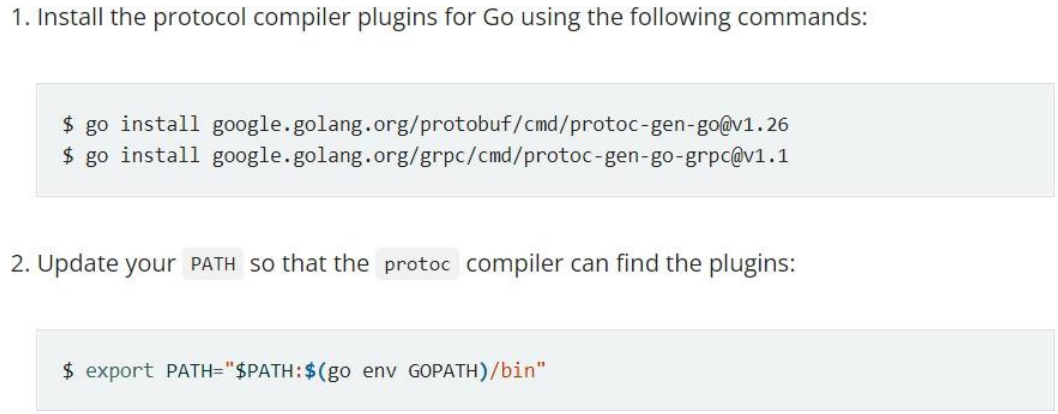
4. Execute the following commands to configure kubectl (also returned by `kubeadm init`).

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

9. Create a helloworld kubernetes file by following the steps in the GitHub page https://github.com/paulbouwer/hello-kubernetes

Exercise 13 Creating a GRPC server-client program and Dockerizing it

1. Install grpc plugins for golang



1. Install the protocol compiler plugins for Go using the following commands:

```
$ go install google.golang.org/protobuf/cmd/protoc-gen-go@v1.26
$ go install google.golang.org/grpc/cmd/protoc-gen-go-grpc@v1.1
```

2. Update your `PATH` so that the `protoc` compiler can find the plugins:

```
$ export PATH="$PATH:$(go env GOPATH)/bin"
```

2. Get the example code

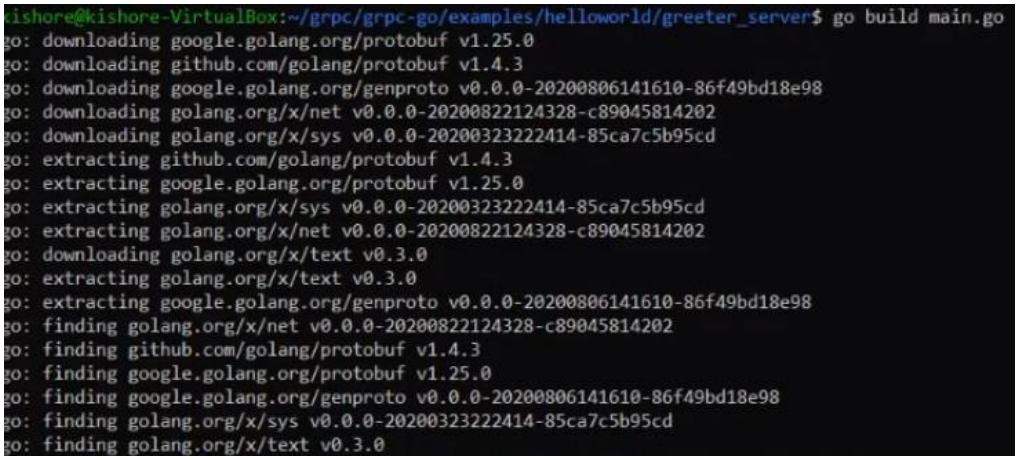
1. Download the repo as a zip file and unzip it, or clone the repo:

```
$ git clone -b v1.41.0 https://github.com/grpc/grpc-go
```

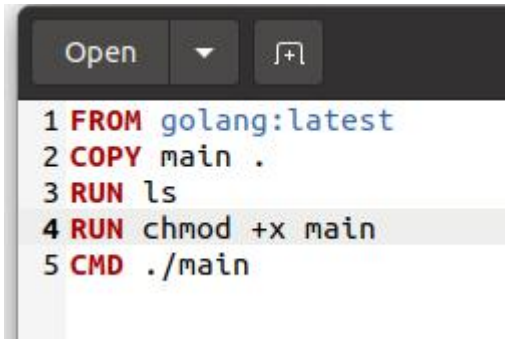
2. Change to the quick start example directory:

```
$ cd grpc-go/examples/helloworld
```

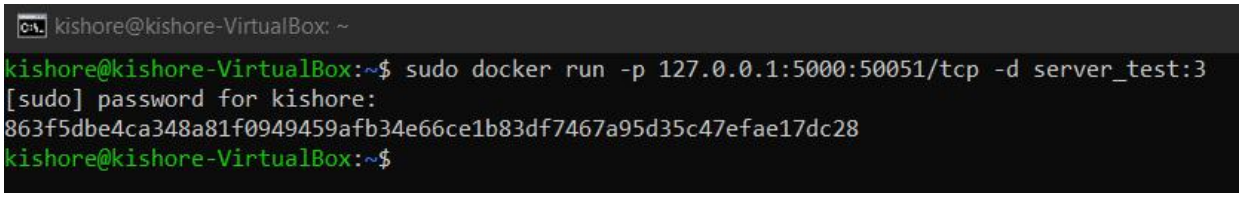
3. Build the main.go file



4. Create a docker file for the grpc server and client program



5. Build both the docker files
6. Change port as 50051 in the greeter go file and enable port forwarding in server using



7. Change the listening port in client as 5000 and run the main.go file

Exercise 14 Containerizing the GRPC server-client program using Kubernetes

1. Create a Kubernetes deployment YAML file from https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
2. Make the necessary changes in the file as below

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grpc-server-client
  labels:
    app: grpc-chat
spec:
  replicas: 3
  selector:
    matchLabels:
      app: grpc-chat
  template:
    metadata:
      labels:
        app: grpc-chat
    spec:
      containers:
      - name: grpc-server
        image: grpc_server:1
        ports:
        - containerPort: 50051
```

3. Create the deployment using

```
kishore@kishore-VirtualBox:~$ kubectl create -f kube.yaml
deployment.apps/grpc-server-client-1 created
kishore@kishore-VirtualBox:~$
```

4. See the deployments using kubectl get deployments

```
kishore@kishore-VirtualBox: ~
kishore@kishore-VirtualBox:~$ kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
grpc-server-client    0/3     3            0           16d
grpc-server-client-1  0/3     3            0           118s
nginx-deployment      0/2     2            0           36d
kishore@kishore-VirtualBox:~$
```

5. Create service.yaml file from https://kubernetes.io/docs/concepts/services-networking/service/

```yaml
apiVersion: v1
kind: Service
metadata:
  name: grpc-servie
spec:
  type: NodePort
  selector:
    app: grpc-chat
  ports:
      # By default and for convenience, the `targetPort` is set to the same value as the `port` field.
    - port: 50051
      targetPort: 50051
      # Optional field
      # By default and for convenience, the Kubernetes control plane will allocate a port from a range (default: 30000-32767)
      nodePort: 30000
```

6. Deploy the services

```
kishore@kishore-VirtualBox:~$ kubectl create -f service.yaml
service/grpc-servie created
kishore@kishore-VirtualBox:~$ kubectl get services
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)           AGE
grpc-servie   NodePort    10.97.45.72    <none>        50051:30000/TCP   13s
kubernetes    ClusterIP   10.96.0.1      <none>        443/TCP           20d
kishore@kishore-VirtualBox:~$
```

7. Change the IP in the greeter_client file and run the program to see the results

```go
package main

import (
        "context"
        "log"
        "os"
        "time"

        "google.golang.org/grpc"
        pb "google.golang.org/grpc/examples/helloworld/helloworld"
)

const (
        address     = "10.0.2.15:30000"
        defaultName = "world"
)
```

```
kishore@kishore-VirtualBox: ~/grpc/grpc-go/examples/helloworld/greeter_client
kishore@kishore-VirtualBox:~/grpc/grpc-go/examples/helloworld/greeter_client$ go run main.go
2021/11/14 15:43:32 Greeting: Hello world
kishore@kishore-VirtualBox:~/grpc/grpc-go/examples/helloworld/greeter_client$
```